# A transformation-driven approach to generate a DSML verification framework

Faiez Zalila, Xavier Crégut, Marc Pantel

# A transformation-driven approach to automate feedback verification results*

Faiez Zalila, Xavier Crégut and Marc Pantel

Université de Toulouse, IRIT – France
Email: firstname.lastname@enseeiht.fr

Authors' version

**Abstract.** The integration of formal verification methods in modeling activities is a key issue to ensure the correctness of complex system design models. In this purpose, the most common approach consists in defining a translational semantics mapping the abstract syntax of the designer dedicated Domain-Specific Modeling Language (DSML) to a formal verification dedicated semantic domain in order to reuse the available powerful verification technologies. Formal verification is thus usually achieved using model transformations. However, the verification results are available in the formal domain which significantly impairs their use by the system designer which is usually not an expert of the formal technologies.
In this paper, we introduce a novel approach based on Higher-Order transformations that analyze and instrument the transformation that expresses the semantics in order to produce traceability data to automatize the back propagation of verification results to the DSML end-user.

**Keywords:** Domain specific modeling language, Formal verification, Model checking, Translational semantics, Traceability, Verification feedback, Fiacre

## 1 Introduction

Model-Driven Engineering (MDE) provides powerful techniques and tools to define *Domain-Specific Modeling Languages* (DSMLs) adapted for given user dedicated domains. These techniques rely on the DSML metamodel that describes the main concepts of the domain and their relations. MDE allows system designers (DSML end-users) working closer to the system domain as they will manipulate concepts from the real system.

Model validation and verification (V&V) activities are key features to assess the conformance of the future system to its behavioral requirements. In order to apply them, it is required to introduce an execution semantics for the DSMLs to verify whether built models behave as expected. This one is usually provided as a mapping from the abstract syntax (metamodel) of the DSML to an existing semantic domain,

* This works was funded by the french Ministry of Industry through the ITEA2 project OPEES and openETCS and the french ANR project GEMOC.

generally a formal verification dedicated language, in order to reuse powerful tools (simulator or model-checker) available for this language [1,2].

One key issue is that system designers are not supposed to have a strong knowledge on formal languages and associated tools. Thus, the challenge for the DSML designer is to leverage formal tools so that the system designer does not need to burden with formal aspects and then to integrate them in traditional Computer Assisted Software Engineering (CASE) tools, like the Eclipse platform.

MDE already provides means to define metamodels (e.g. Ecore tools, Emphatic) along with static properties (e.g. OCL) and to generate either textual syntactic editors (e.g. xText, EMFText) or graphical editors (e.g. GMF, Spray, Sirius). Additionally, the DSML designer should extend the DSML framework with required elements to perform V&V tasks relying on translational semantics from the DSML to formal languages (e.g. Petri nets, automata, etc.).

Translational semantics for DSMLs into formal semantic domains allows the use of advanced analysis tools like model-checkers. It introduces the executability aspect for DSMLs and can provide execution paths in case of verification failures. However, this approach has a strong drawback: the verification results are generated in the formal technical space, whereas DSML end-users are not supposed to have a strong knowledge on formal languages and associated tools. Therefore, these results should be lifted to the user level (i.e. the DSML level) automatically.

The DSML designer does not only provide the translational semantics but must also implement a backward transformation that bring back the formal verification results to the DSML level so that they are understandable by the system designer (DSML end-user). Bringing back formal results to the DSML level can be complex. It is thus mandatory to assist the DSML designer by providing methods and tools to ease the implementation of result back propagation.

In this work, we propose a generic approach to integrate hidden formal verification through model-checking for a DSML. We rely on the Executable DSML pattern [3] to define all concerns involved in the definition of DSML semantics while its translational semantics is defined using a model-to-model transformation. Then, we define a higher-order transformation (HOT) to manipulate the translational semantics in order to generate the mandatory tools to ensure the back propagation of verification results. We rely on the Fiacre intermediate language [4] to help DSML designers to use formal methods by reducing the semantic gap between DSMLs and formal methods.

To illustrate our proposal, we consider as running example the xSPEM executable extension of the SPEM process modeling language [5]. It was designed in order to experiment V&V in the TopCased toolkit [6] using an MDE approach.

The paper is structured as follows: section 2 presents the *Executable DSML pattern* illustrated with SPEM [5] and the Fiacre metamodel. Section 3 explains our approach to generate the DSML verification framework. Section 4 presents the overview of the use of the generated DSML verification framework by the system designer. Section 5 gives some related work in the domain of user level verification results. Finally, Section 6 concludes and gives some perspectives.

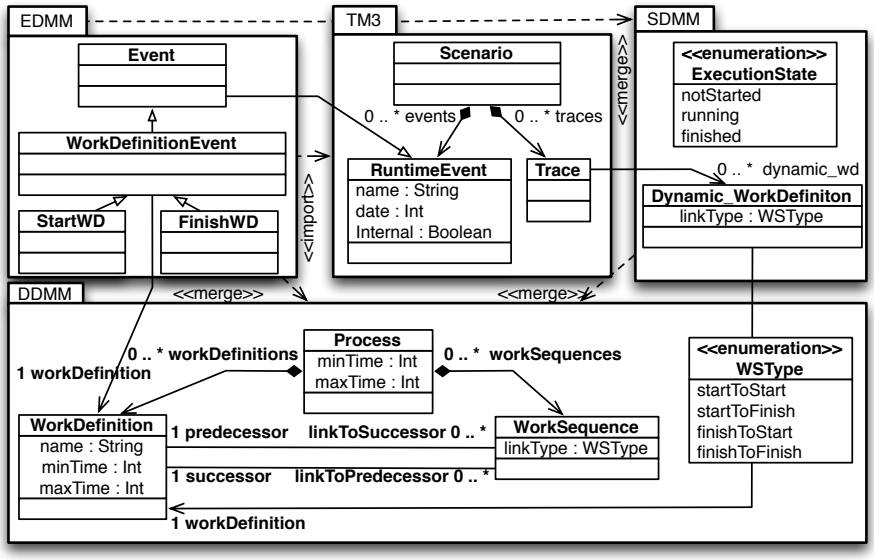


**Fig. 1.** *Executable DSML pattern* applied into the SPEM metamodel

## 2 The *Executable DSML pattern* for V&V

When dealing with an executable DSML, the usual metamodel generally does not model the notions manipulated at runtime such as dynamic information or stimuli that make the model evolve. To address this issue, Combemale et al. have proposed the *Executable DSML pattern* [3]. In the context of formal verification, the pattern is applied both on the source DSML and the formal target DSML to help in leveraging to the DSML side the results obtained on the formal side.

### 2.1 xSPEM as an Executable DSML

The *Executable DSML pattern* applied on SPEM metamodel is shown in Figure 1.

The classical DSML metamodel is shown as the *Domain Definition MetaModel (DDMM)*. It provides the key concepts of the language (representing the considered domain) and their relationships. It is the usual metamodel used to define the modeling language in standardization organisations. It is usually endowed with structural constraints. For instance, the SPEM metamodel defines the concepts of *Process* composed of (1) *workDefinitions* that model the activities performed during the process, (2) *workSequences* that define temporal dependency relations (causality constraints) between activities.

During the execution of a model, additional data are usually mandatory for expressing the execution itself. A first extension, named *State Definition MetaModel (SDMM)*, stores dynamic data in the form of metatype instances. For example, each workdefinition is in one of the states: *notStarted, running or finished*. The state

attribute should be defined in the *Dynamic_WorkDefinition* metatype. The second extension, *Event Definition MetaModel (EDMM)*, reifies the concrete stimuli of the DSML as subtypes of the common abstract RuntimeEvent metatype. Concrete EDMM events add properties in relation to, and/or redefine properties of, events related to the formal semantics to be supported. As an illustration, runtime events for xSPEM include "start a workdefinition" and "finish a workdefinition". Thus, two metatypes *StartWD* and *FinishWD* will be defined.

Finally, the *Trace Management MetaModel (TM3)* allows to define a scenario as a sequence of runtime events usually interleaved with the state of the model between triggered previous and next events. The TM3 is independent of any DSML.

### 2.2 The Fiacre formal language

Fiacre [4] is a french acronym for an *Intermediate Format for Embedded Distributed Components Architectures*. It was designed as the target language for model transformations from different DSMLs such as Architecture Analysis and Design Language (AADL), Ladder Diagram (LD), Business Process Execution Language (BEPL) and some UML diagrams (sub-languages).

The Fiacre formal language allows representing both the behavioral and timing aspects of systems, in particular embedded and distributed systems, for formal verification and simulation purposes. Fiacre is built around two notions:

- Processes describe the behavior of sequential components. A process is defined by a set of control states, each associated with a piece of program built from deterministic constructs available in classical programming languages (assignments, conditionals, repetitions, and sequential compositions), non deterministic constructs (non deterministic choice and non deterministic assignments), communication events on ports, and transitions to next state.
- Components describe the composition of processes, possibly in a hierarchical manner. A component is defined as a parallel composition of instantiated components and/or processes communicating through ports and shared variables. The notion of component also allows restricting the access mode and visibility of shared variables and ports, associating timing constraints with communications, and defining priority between communication events.

Verification results are obtained at the formal level and must be leveraged at the DSML level. This feedback is made easier thanks to the *Executable DSML pattern* [3] applied not only at the DSML level but also at the formal one.

In this work, our aim is to build DSML events out of the Fiacre ones provided by verification failures. We are thus interested only in the Fiacre EDMM. It contains specific events [7]: instances of processes entering or leaving a state, variables changing values, communications through ports and tagged statements occurring in process instances.

## 3 Generation DSML verification framework

Our approach is based on two model-based concepts: 1) **traceability** and 2) **higher-order transformation**. It relies on the Atlas Transformation Language (ATL) [8].

### 3.1 Traceability mechanism

The model transformation traceability consists in storing a set of relations (also named mappings) between corresponding source and target model elements in order to reuse them to verify and validate software life-cycle.

Several traceability approaches are proposed in the literature [9]. For example, in [10], authors introduce an approach named embedded traceability. In this latter, the traceability elements are embedded inside the target models. For [11], the traceability information are considered as a model, more precisely as an additional target model of a transformation program. We have chosen the last because it avoids polluting source and target models with traceability information. Traceability information are generated while running the model transformation (*Source2Target model transformation*) as illustrated in Fig. 2.

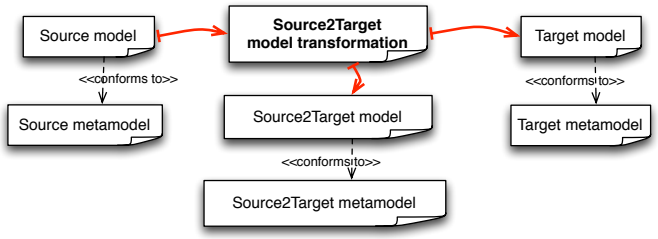


**Fig. 2.** The traceability mechanism

### 3.2 Higher-Order Transformation

A higher-order transformation is a model transformation that manipulates other model transformations. It means that the input and/or output models are themselves model transformations. Fig. 3 gives a technical overview of the application of a higher-order transformation where both inputs and outputs are model transformations [12]. A first step parses the textual syntax and builds a model conforming to the ATL metamodel. Then, the higher-order transformation (*ATL Higher-Order transformation*) manipulates the input model and generates another transformation model (*ATL output model*). It generally adds some information. We use it to easily extend the translational semantics to support traceability data generation. Finally, the textual representation (*ATL output transformation*) was generated from the generated ATL model.

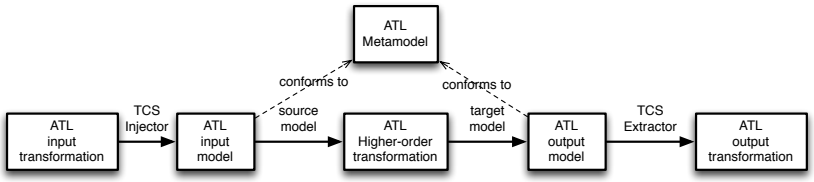


**Fig. 3.** An overview of Higher-Order transformation

### 3.3 The approach architecture

The figure 4 shows the overall organization of our approach. It explains different steps performed by the DSML designer in order to prepare the verification framework used by the DSML end-user. It also introduces the capability of our approach to simplify the DSML designer task in order to facilitate the integration of a verification framework for a new DSML.

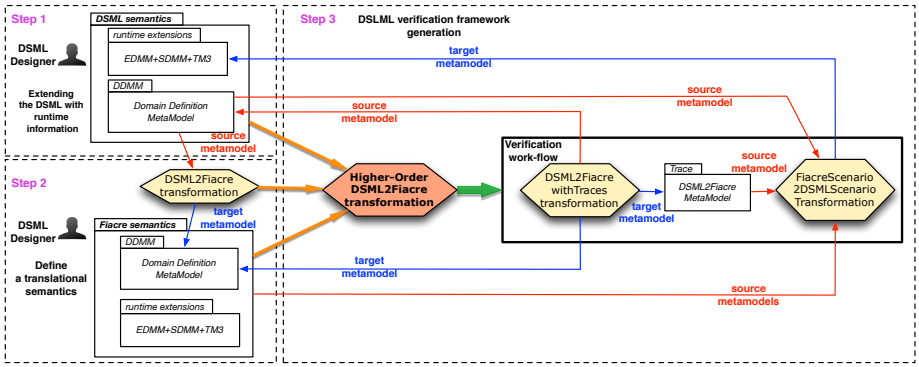


**Fig. 4.** Scheme of different steps to generate the DSML verification framework

**Step 1.** The DSML designer extends the DSML abstract syntax (DDMM) with the runtime information conforming to the *Executable DSML pattern*. It consists in defining different events (EDMM) and snapshots (SDMM) to be captured.

**Step 2.** The DSML designer defines the translational semantics (*DSML2Fiacre transformation*) from DSML DDMM into the formal FIACRE language. In our case, it is defined as a model-to-model transformation implemented with ATL.

In our example, the translational semantics consists in transforming a SPEM model into a FIACRE specification. Here are some rationale behind this translational semantics. Each workdefinition is translated into one FIACRE process. Such a process contains three states (*notStarted*, *running* and *finished*) and two transitions (from

*notStarted* to *running* and then from *running* to *finished*). The transitions are guarded (conditional statement) according to the dependencies defined between workdefinitions (the previous activities must have reached the expected state). As a FIACRE process cannot inspect the current state of other processes, the process takes as argument an array containing the state of each workdefinition (derived from the xSPEM SDMM). Each transition includes an assignment to update variables which store the state of the activities.

In addition, the transformation must integrate the information to capture the DSML events. FIACRE allows to annotate the FIACRE model with tag statements "#ident". So, the DSML designer must extend the translational semantics with tag statements in order to capture the corresponding event in the DSML scenario.

The listing 1.1 shows an application of this extension for the *StartWD* event which is a workdefinition event. A tag declaration named *"StartWD"* is defined (lines 12-13) in order to capture the *StartWD* event defined in the SPEM EDMM. A tag statement is initialized with the corresponding tag declaration (lines 9-10). This statement is inserted in the first FIACRE transition (*from notStarted ... to running*) just before the assignment which updates the state variable of the activity (lines 6-7). These elements are also defined for the *FinishWD* event.

```
1  rule WorkDefinition2Fiacre {
2    from
3      workdefinition: SpemMetaModel!WorkDefinition
4    to
5     ...
6    sequence_statement_start: FiacreMetaModel!StatementSequence(
7      statements <- Sequence{start_tag_statement, assignment_is_started}
8    ),
9    start_tag_statement: FiacreMetaModel!TaggedStatement(
10     tag <- tag_start_wd
11   ),
12   tag_start_wd: FiacreMetaModel!TagDeclaration(
13     name <- 'StartWD'
14   )
15    ...
```

**Listing 1.1.** Extending the translational semantics with tagged statements

**Step 3.** The *DSML2Fiacre transformation* defined by the DSML designer provides the proposed translational semantics extended with bindings which allow implementing DSML events. Based on this model-to-model transformation, a higher-order transformation (*Higher-Order DSML2Fiacre transformation*) allows to generate different elements to extend the DSML verification framework. Let's detail the generated elements.

**DSML2Fiacre metamodel** : It is the traceability metamodel. It defines a metaclass for each traceable element.

**Definition 1** *A traceable element is a* DSML *element extended at least with an event in the* DSML EDMM.

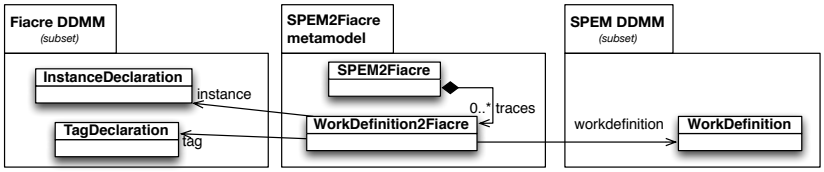


**Fig. 5.** SPEM2Fiacre trace metamodel

The generated meta-class contains references to record the traceable element (*workdefinition*), the tag declaration (*tag*) and its corresponding process instance (*instance*). In the case of xSPEM, only *WorkDefinition* is a traceable element. The generated trace metamodel, SPEM2Fiacre, is shown in Figure 5.

Therefore, a meta-class named *WorkDefinition2Fiacre* was generated to store the workdefinition and the corresponding tag event containing the tag declaration and its instance. The traceability metamodel is generated in order to be referenced by the translational semantics enriched with traceability rules.

**DSML2FiacrePlusTraces transformation** : It adds traceability rules to the translational semantics proposed in the *DSML2Fiacre transformation*. Traceability elements are created into the rule which translates the traceable element.

Typically, the traceability information consists in saving a tag event information (FIACRE side) and the traceable element (DSML side).

Listing 1.2 introduces elements into the *WorkDefinition2Fiacre* rule (line 1). According to the generated *DSML2Fiacre* metamodel, this rule produces a target pattern element (lines 6-9) to create *WorkDefinition2Fiacre* instance (line 6) with the tag declaration (line 7) and the generated FIACRE process instance (line 8). Saving traceability source information in ATL consists in adding a single-statement imperative block to initialize the source element, *workdefinition*, with the traceable element (lines 10-13).

```
1  rule WorkDefinition2Fiacre{
2      from
3          workdefinition: SpemMetaModel!WorkDefinition
4      to
5       ...
6       start_trace : SpemMetaModel2FiacreMetaModel!WorkDefinition2Fiacre (
7           tag <- tag_start_wd,
8           instance <- process_instance
9       )
10      do {
11          start_trace.refSetValue('workdefinition', workdefinition);
12          ...
13      }
14      ...
```

**Listing 1.2.** The generation of traceability target pattern elements

**FiacreTM32SpemTM3 transformation** : It defines the backward transformation which allows to back propagate the FIACRE formal scenario into the DSML scenario. Listing 1.3 shows some elements in this ATL transformation.

It defines an ATL helper (lines 1-5) for each traceable element (*WorkDefinition* in this case) to request from a tag event the corresponding trace element (*WorkDefinition2Fiacre*). An ATL rule (lines 7-14) is defined for each DSML event to generate from a tag event (lines 8-10) using the trace element a corresponding DSML event (lines 11-13).

```
1  helper context FiacreSemanticsMetaModel!"fiacreSemantics::fiacreEDMM::TagEvent"
2  def: getTraceabilityElement() :
3       Spem2FiacreMetaModel!WorkDefinition2Fiacre =
4       Spem2FiacreMetaModel!WorkDefinition2Fiacre.allInstances()
5       ->select(trace|trace.tag=self.tag and trace.instance=self.instance)->first();
6
7  rule TagEvent2StartEvent{
8   from fcr_event:
9     FiacreSemanticsMetaModel!"fiacreSemantics::fiacreEDMM::TagEvent"
10      (fcr_event.tag.name='StartWD')
11  to spem_event:
12    SpemSemanticsMetaModel!"spemSemantics::spemEDMM::StartWD"(
13          workdefinition <- fcr_event.getTraceabilityElement().workdefinition)
14 }
```

**Listing 1.3.** A subset of the backward transformation

## 4 The use of the DSML verification framework

Once the previous steps have been performed, the DSML verification framework is generated. Figure 6 shows an overview of the generated DSML verification framework connected to both modeling and formal levels.

This framework allows the DSML end-user to define models (xSPEM *model*) conforming to the DSML abstract syntax (xSPEM DDMM) and to verify behavioral properties while hiding formal methods and tools.

The defined model is translated with *SPEM2FiacrewithTraces* transformation into a FIACRE program (*fiacre model*). Additionally, a traceability model (*spem2fiacre* model) is also generated which saves mappings between both models.

Next, the existing tools around the FIACRE language (FRAC compiler[1] and SELT[2], the TINA [13] model-checker) perform the formal verification and generate the formal results (*counter-example* in case of failure). In the same spirit defined in this paper, we have transformed these formal results at the Petri nets level to more abstract results at the FIACRE level (*fiacre scenario*) [14]. Finally, the backward transformation (*FiacreScenario2SPEMScenario* transformation) is performed in order to generate the expected scenario (xSPEM *scenario*).

[1] `http://projects.laas.fr/fiacre/manuals/frac.html`
[2] `http://projects.laas.fr/tina/manuals/selt.html`

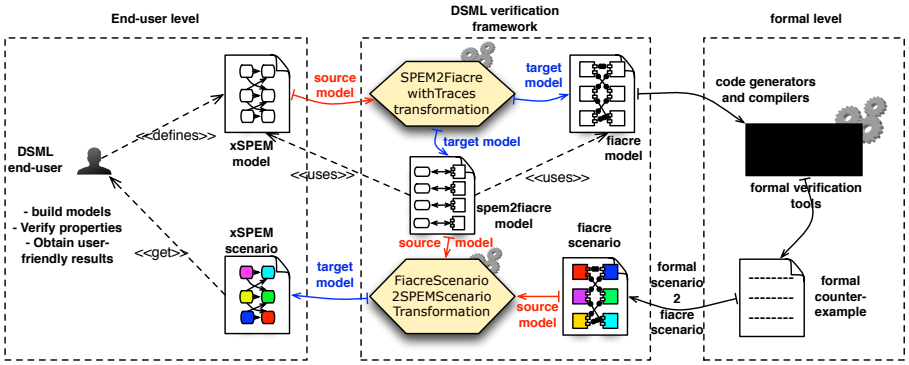


**Fig. 6.** The use of the generated DSML verification framework

## 5 Related Work

The problem of integrating formal verification into the design of DSMLs has been widely addressed by the MDE community. However, the analysis feedback at the DSML level problem is typically either ignored or resolved by defining ad-hoc or hard-coded solutions. For example, in [15], authors propose an approach, named *Metaviz*, based on the real-time systems specification and validation tool set IFx-OMEGA. It is designed to ease the visualization of simulation trace. The goal is to assist the user in the Interactive Simulation task by refining this step with a diagnosis process built around visualization concepts. It consists in feeding back verification results at OMEGA level. Thus, It can be considered as an ad-hoc approach.

On the other hand, a few number of works handling the feedback with general solutions exists in the literature.

Hegedüs et al. [16] propose a technique for the back propagation of simulation traces based on change-driven model transformations from traces generated by SAL model checking framework to the specific animator named BPEL Animation Controller. So, they define a change-driven model transformation which consumes changes of the Petri nets simulation run and produces a BPEL process execution using traceability information generated while running the translational semantics defined previously. In this case, after defining the runtime extension for both levels (BPEL and Petri nets) and the translational semantics, the DSML designer is invited to define 1) a change command metamodel for Petri nets and BPEL and also 2) the back-ward change-driven transformation. In our approach, we try to generate automatically the mandatory data required to feedback verification results without introducing additional information.

In [17], authors introduce an algorithm requiring the DSML's semantics to be defined formally, and a relation R to be defined between states of the DSML and states of the target language. The DSML designer must provide as input a natural-number bound $n$, which estimates a difference of granularity between the semantics

of the DSML and the semantics of the target language. However, we don't think that DSML designer, for who it is difficult to use formal methods and verification, can define this important information to feedback verification results.

The most advanced work about back propagation problem is defined in [18]. A *Triple Graphical Pattern (TGP)* is defined to introduce how a result generated in the formal domain can be shown in the DSML level. TGP is defined to resolve the problem of 1-to-1 restriction on back propagation. It allows to handle the 1-to-n case which means that several events in the formal verification results may correspond to one event in the DSML level and m-to-n case which considers mappings as a set of events in the both levels. These cases occur due to the mismatch between trace granularity between the DSML and formal levels caused by the semantic gap between both levels. In our case, the use of the intermediate language FIACRE allows to reduce this semantic gap and therefore the 1-to-n and m-to-n mapping are usually not occurring. Also, the DSML designer does not need to extend the translational semantics with any additional information to ensure the back-annotation task. Finally, as a technical viewpoint, defining a Triple Graph Transformation Systems (TGTS) is typically more complicate task than an ATL transformation.

## 6 Conclusion

In this paper, we have presented an approach to integrate formal tools into the verification of DSMLs. It consists in generating a DSML verification framework containing the necessary elements to map the DSML abstract syntax into a semantic domain and also to feed verification results — generated in the formal level — back to the DSML level.

Our solution is a generic tool implementing a higher-order transformation that requires a translational semantics defined between the DSML and the FIACRE intermediate language. The translational semantics is extended with tagged statements which trigger DSML events. This additional information allows identifying which elements are concerned with the back propagation task. Therefore, all required elements are generated automatically. It has been illustrated on xSPEM as DSML and FIACRE as the formal language.

This approach has been designed for domain specific languages. It is currently being experimented for several significantly different DSMLs like Architecture Analysis and Design Language (AADL), Business Process Engineering Language (BPEL) and Ladder Diagram (LD).

## References


1. J. Merilinna and J. Pärssinen, "Verification and validation in the context of domain-specific modelling," in *Proceedings of the 10th Workshop on Domain-Specific Modeling*, ser. DSM '10. New York, NY, USA: ACM, 2010, pp. 9:1–9:6.
2. D. Harel and B. Rumpe, "Meaningful Modeling: What's the Semantics of "Semantics"?" *Computer*, vol. 37, no. 10, pp. 64–72, 2004.

3. B. Combemale, X. Crégut, and M. Pantel, "A Design Pattern to Build Executable DSMLs and associated V&V tools (short paper)," in *Asia-Pacific Software Engineering Conference (APSEC), Hong Kong, China*, 2012.
4. B. Berthomieu, J.-P. Bodeveix, M. Filali, P. Farail, P. Gaufillet, H. Garavel, and F. Lang, "FIACRE: an Intermediate Language for Model Verification in the TOPCASED Environment," in *ERTS'08*, Jan. 2008.
5. *Software & Systems Process Engineering Metamodel (SPEM) 2.0* , Object Management Group, Inc., Oct. 2007.
6. P. Farail, P. Gaufillet, A. Canals, C. L. Camus, D. Sciamma, P. Michel, X. Crgut, and M. Pantel, "The TOPCASED project: a Toolkit in OPen source for Critical Aeronautic SystEms Design," in *Embedded Real Time Software (ERTS'06)*, Toulouse, 25-27 January 2006.
7. N. Abid and S. Dal Zilio, "Real-time Extensions for the Fiacre modeling language," pp. http://automata.rwth–aachen.de/movep2010/index.php?page=about, 2010.
8. F. Jouault and I. Kurtev, "Transforming Models with ATL," in *Satellite Events at the MoDELS 2005 Conference, Proceedings of the Model Transformations in Practice Workshop*, ser. LNCS, vol. 3844. Springer, 2005.
9. I. Galvao and A. Goknil, "Survey of traceability approaches in model-driven engineering," in *Enterprise Distributed Object Computing Conference (EDOC). 11th IEEE International*, Oct. 2007, p. 313.
10. D. S. Kolovos, R. F. Paige, and F. A. Polack, "Merging models with the epsilon merging language (eml)," in *Model Driven Engineering Languages and Systems*. Springer, 2006, pp. 215–229.
11. F. Jouault, "Loosely coupled traceability for atl," in *In Proceedings of the European Conference on Model Driven Architecture (ECMDA) workshop on traceability*, 2005.
12. M. Tisi, F. Jouault, P. Fraternali, S. Ceri, and J. Bézivin, "On the use of higher-order model transformations," in *Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications*, ser. ECMDA-FA '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 18–33.
13. B. Berthomieu, P.-O. Ribet, and F. Vernadat, "The tool TINA – construction of abstract state spaces for Petri nets and time Petri nets," *Int. Journal of Production Research*, vol. 42, no. 14, pp. 2741–2756, 2004.
14. F. Zalila, X. Crégut, and M. Pantel, "Verification results feedback for FIACRE intermediate language," in *Confrence en Ingnierie du Logiciel (CIEL)*, Jun. 2012. [Online]. Available: http://gpl2012.irisa.fr/?q=node/31
15. I. Ober, I. Ober *et al.*, "Seeing errors: model driven simulation trace visualization," in *Model Driven Engineering Languages and Systems*. Springer, 2012, pp. 480–496.
16. Á. Hegedüs, G. Bergmann, I. Ráth, and D. Varró, "Back-annotation of simulation traces with change-driven model transformations," in *SEFM'10*, 2010, pp. 145–155.
17. B. Combemale, L. Gonnord, and V. Rusu, "A generic tool for tracing executions back to a dsml's operational semantics," in *ECMFA*, 2011, pp. 35–51.
18. E. Guerra, J. de Lara, A. Malizia, and P. Díaz, "Supporting user-oriented analysis for multi-view domain-specific visual languages," *Information and Software Technology*, vol. 51, no. 4, pp. 769–784, 2009.